TQHT : A Tool for Querying HTML Documents in Intranets

by

KAPIL MOHAN

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

FEBRUARY 1997

TQHT: A Tool for Querying HTML Documents in Intranets

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

Kapil Mohan

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

Feb 1997

CSE-1997-M-MOH-TQHT

CERTIFICATE

This is to certify that the work contained in the thesis entitled TQHT: A Tool for Querying HTML Documents in Intranets by Kapil Mohan has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

[illegible]

T. V. Prabhakar,
Associate Professor,
Department of Computer Science & Engineering,
Indian Institute of Technology, Kanpur.

Abstract

In the present work we have designed and implemented TQHT, a tool for searching HTML documents in an intranet. TQHT can be used in two ways: as a search engine or as a dynamic link. Both require a powerful query language, which is supported by TQHT. Main features of this query language are:

- Queries can be based on content as well as structure of the document.
- Portions of a document can be retrieved(displayed).
- No pre-indexing of the documents is required.

These features are supported by searching the document set exhaustively. This approach is applicable for small search domains like intranets or a small number of sites on the Internet.

Acknowledgments

I wish to express my sincere gratitude to Dr. T. V. Prabhakar for giving me an interesting problem to work on and for his constant guidance throughout the work. His ideas and discussions have been exciting to me. Working under him has been a memorable experience.

I take the opportunity to thank the class of mtech95 which has made my stay here, a joyful and memorable experience. I would like to give special thanks to my friends Alkesh, Deepak and Nigam for helping me whenever I needed.

I am grateful to my parents & brother for serving as the constant source of motivation

Contents

List of Figures

Chapter 1

Introduction

Intranets have emerged as a powerful tool for in-house information processing needs of enterprises. Intranet is integration of Internet and web capabilities into a local area network. The advantages of intranets over traditional approaches are:

- connected computers can be heterogeneous,
- hypertext model simplifies, information retrieval and navigation,
- large number of applications available for Internet such as, web browsers [10,11], search engines[14], HTML authoring tools[13], etc. can be used for information management.

A very important capability of the web is its data model. Information on the web is structured in the form of a network, where each node of the network is a hypertext document, and each edge of the network is a hypertext link. User can navigate in the web of information by just clicking the links in the documents. Documents are formatted using HTML (Hyper Text Mark-up Language[12]), which allows users to write documents in hypertext format.

In the remaining part of this chapter we will discuss about World Wide Web, hypertext and HTML in more detail. We will also discuss the motivation behind our work and in the end thesis organization is presented.

1.1 World Wide Web

The World Wide Web(WWW) is todays most popular and advanced information system. The World Wide Web is a distributed multimedia hypertext system. Distributed means information on the web can be located on computer systems around the world, multimedia means this information can include text, graphics, sound or video. Hypertext means access to the information is available using hypertext techniques, which typically involve using a mouse to select a highlighted image or phrase. After selecting the phrase or image, information relevant to this is retrieved. This information can be retrieved from any part of the world.

The World Wide Web was initially developed to provide an infrastructure for particle physicists throughout Europe to share information[20]. Since physicists had a variety of computer systems, so one of the major issues in the development was portability. The World Wide Web was developed using the client-server architecture, which ensures the cross platform portability.

1.1.1 Client Server Architecture

The World Wide Web is based on the client server architecture which is shown in the figure 1.1. The end user accesses the World Wide Web through a client process, known as browser. The client will display hypertext links in some manner, such as underlining the links. By selecting a link a request is sent over the network to a World Wide Web server which typically runs on a powerful computer system. The server will retrieve the file which has been requested and deliver it to the client. In different types of browsers procedure of link selection is different. In graphical browsers, link is selected by clicking the mouse button; in text-based browsers, link is selected by typing some number.

Once the client has started to retrieve the file it can display it on the local machine. If the client can not display the file, it may pass the file to a external viewer which can display the file.

Principle of the WWW is that once information is available on the web, it should be accessible from any type of computer, located any where in the world by an

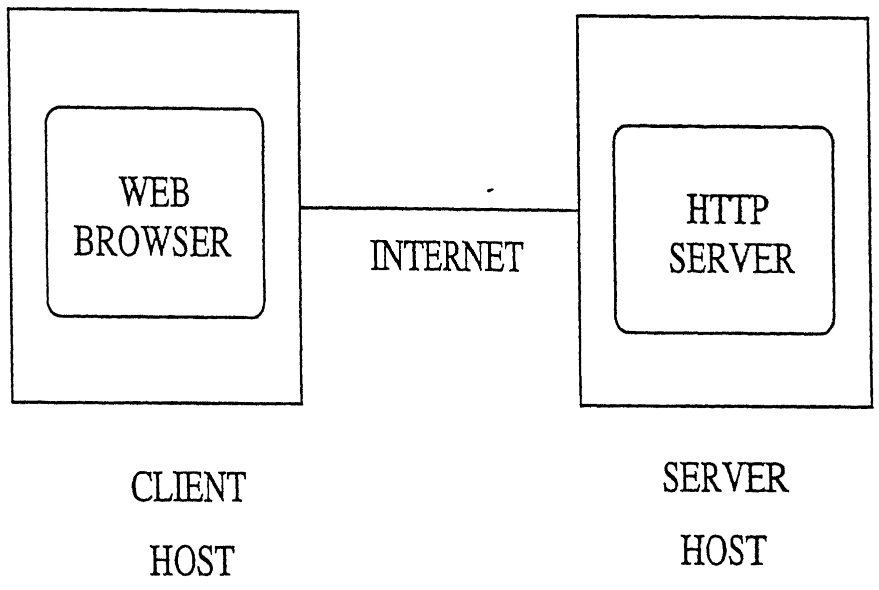


Figure 1.1: Client server architecture of the web

authorized person using a simple interface program.

1.1.2 Web browsers

To access information on the WWW, one needs to use browser applications that lead the user to the desired documents, and then display that information. Browsers access files using FTP, HTTP, Gopher and several other protocols. Browser clients can also search remote documents and databases if the associated server at that site has built in search capabilities.

For conventional vt220 terminals line-oriented browsers can be used, while for workstations graphical browsers can also be used. Lynx is an example of line-oriented browsers, while Netscape, Mosaic etc. are the examples of graphical browsers.

1.1.3 Uniform Resource Locator

Each document on the web can be identified by a unique address, which is known as Uniform Resource Locator(URL). Structure of the URL is:

```
protocol://host_address[:port]/path/file_name
```

where protocol is one of the following:

- **file** - a document on the local system or a file on an anonymous FTP server.
- **http** - a document on a world wide web server
- **gopher** - a document on the gopher server.
- **wais** - a document on the wais server.
- **news** - a Usenet newsgroup.
- **telnet** - a connection to a telnet based service.

Other parts of the URL are:

- **host_address** - is the address of the server,
- **port** - is port number of the http server,
- **path** - gives the location of the file, and
- **file_name** - gives the name of the file.

For example a sample URL is:

```
http://www.w3.org/hypertext/Products/wais/sources/Overview.html
```

In the above URL:

- **protocol** is http,
- **hostname** is www.w3.org,
- **path** is /hypertext/Products/wais/sources, and
- **file_name** is Overview.html.

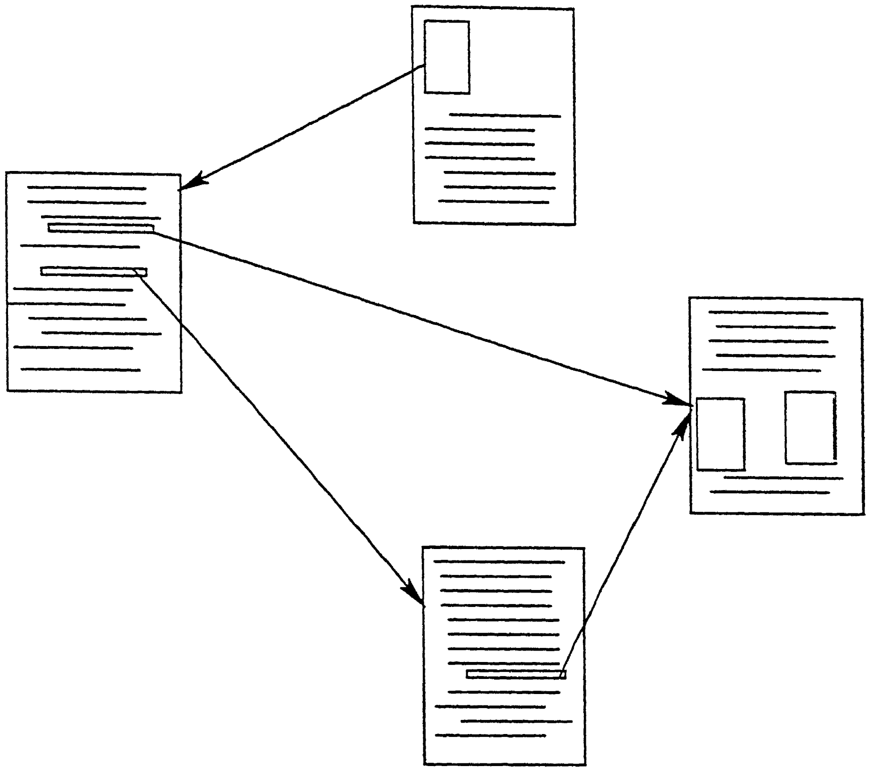


Figure 1.2: A hypertext network

1.2 Hypertext

Hypertext is a text with links. The Hypertext is not a new idea. In fact when we read a book, we can see references to other books or references to some other part of the same book. With Hypertext, the computer makes following such references as easy as turning a page. User is required to click a mouse button and the referred information will be there on the screen. It allows the reader to escape from the sequential organization of the papers to follow a thread of his own interest.

Hypertext is all about connectivity among documents. World Wide Web uses hypertext as a method of presentation. In WWW links can lead from whole or part of document to whole or part of another document. Besides textual documents, graphical or sound documents are also supported.

Major advantages of hypertext model[15]:

Matches with human thinking - One can easily navigate through hypertext, since the hypertext model is similar to model of human thinking.

Ease of tracing references - Machine support for link tracing means that all references are equally easy to follow forward to their referent, or backward to their reference.

Ease of creating new References - User can grow on their own networks or simply annotate some one else's document with a comment, with out changing the referred document.

Information structuring - Both hierarchical and non hierarchical organization can be imposed on unstructured information.

Customized documents - Information segments can be threaded together in many different ways allowing the same document to serve multiple purpose.

Modularity of information - Since the same information can be referenced from several places, ideas can be expressed with less overlap and duplication.

Task scheduling - The reader is supported in having several paths of inquiry active and displayed on the screen at the same time, such that any given path can be unwound to the original task.

Collaboration - Several authors can collaborate with the document and comments about the document can be tightly interwoven so as to create a collaborative environment.

Disadvantages of using hypertext[15]:

Disorientation - If the network of nodes and links is very large, the reader may face the problem of disorientation. During the navigation of hyperspace, some confusion may arise regarding the current position in the network as a whole and the path following which some particular place in the network can be reached. The problem of losing orientation occurs while going through the linear text also but is more pronounced in the hypertext because a hypertext system provides greater degree of freedom for traversal.

Cognitive overhead - The other problem with using hypertext is that it is difficult to become accustomated to the additional mental effort and concentration required by the author to create, name and keep track of links.

1.2.1 Dynamic Links

The power of a hypertext document is its capability of having links to other documents. There are three types of links possible in hypertext: static, partially dynamic and fully dynamic[3,4]. Static links are those links which have a fixed target and destination of the link. These are fixed at the time of link creation. Second type of links are partially dynamic links, these type of links have a fixed source, but the destination of the link is not fixed. Destination of the link is evaluated according to some pre-specified rules, at the time of traversing the link. Third type of links are fully dynamic links. Target as well as the source of these type of links are evaluated at the time of traversing the link.

For a small hypertext system whose content and location does not change frequently, static links have sufficient capabilities. One can update the links manually according to changes, which seldom happens to occur. But if the hypertext system is large and hypertext nodes change frequently, managing static link is not possible. For these systems dynamic links can be used. Dynamic links provide a system with following capabilities.

- The system can automatically assimilate new components using the existing link specification. It can also handle changes in the content and location of the documents, automatically.

- Information retrieval requirements keep changing. System can be easily coustomized according to the changing requirements.

1.3 HTML

HTML stands for Hypertext Markup Language[12]. It is a simple text based markup language for creating hypertext documents which can be read by browsers such as Netscape and Mosaic. HTML is a Document Type Definition(DTD) of Standard Generalized Mark-up Language(SGML). Author of a HTML document defines the structure, layout, hypertext links in the document using HTML tags. HTML tag consists of a left angle bracket(<), a tag name, and a right angle bracket(>). Tags are usually paired(e.g.,<H1> and </H1>) to start and end the tag instruction. The end tag looks just like the start tag except a slash(/) precedes the text within the brackets.

The HTML language was first specified in 1991 by Tim Berners-lee of CERN as part of WWW initiative. The specification continued to evolve to meet the requirements of the web users. In 1993, Dan Connelley wrote the first SGML DTD describing HTML. Presently World Wide Web consortium leads and coordinates the development of HTML. Latest version of HTML is 3.2.

1.4 Motivation and Objectives

In a small hypertext network user can locate the required information by navigating through the web, but for larger hypertext networks searching information becomes difficult. Search engines help a user in locating information on the net. A large number of search engines exist on the Internet, but most of these search engines deliver a high degree of irrelevant information when user gives some complex query. One reason for this problem is that, most of the search engines support a very primitive query language; complex queries can not be formulated effectively. And further these search engines support only content based queries, structure of the documents can not be specified. One can specify what words should be present in the document, but one can not specify the position of these words in the documents. For example, a user may want to see all the documents which have the words *java* and *introduction* in **h1** level header. Such a query can not be specified unless one can address the structure of the document. Query languages based on content as well as structure of the document can be more expressive and powerful[1].

Another problem with traditional search engines is that, as a result of the query whole document is returned, even though a part of the document is relevant to the query. Some times user may be looking for some specific parts of the documents and he may not be interested in the whole document. For example some times user may want to get only that para of the document which contains some specific word.

There is a problem with HTML documents, it is limited capabilities of links. HTML documents can have static links only, which are fixed at the time of link creation. Static links work fine if the documents' content and location change seldom. We can update the links manually. But manual updation is not possible if documents change frequently and are large in number. One solution to this problem is dynamic links[2,3,4,6]. Target of a dynamic link is fixed at the time of traversing by the system according to some pre-specified rules.

As the Intranets are becoming more and more popular, we were motivated to develop a tool for searching the HTML document in the intranet environment which could overcome the above mentioned limitations. This tool would be able to support content and structure based queries as well as allow a user to specify a part of the document. This tool can also work as a dynamic link.

1.5 Thesis Organization

In chapter 2 we introduce traditional search engines. In Chapter 3 we have presented design of TQHT: A Tool for Querying HTML documents in intranets. Chapter 4 gives implementation details of TQHT. In chapter 5 conclusions and future work are discussed. Appendix 1 presents some query examples. In Appendix 2 the user manual for using TQHT is given.

Chapter 2

Traditional Search Engines

Information on the web is so huge that it is difficult to locate required information. A search engine is a tool which is specially designed for this purpose. We simply enter key-words, and the search engine takes over. There are two major parts of this whole process of searching. One is communication between user and the search engine and another is process of searching. Communication between the user and the search engine includes receiving the query from the user, transferring this query to the search engine and transferring the search results from the search engine to the user. Second part includes searching the indexed documents for finding out the results. Search engine is an external application to the WWW server which is hooked up by the http server using CGI.

2.1 CGI

The Common Gateway Interface(CGI)[19] is a standard for interfacing external applications with information servers, such as HTTP or web servers. For example, if someone wants a search engine to hook up to the web then he is required to create a CGI program that the Web daemon will execute to transmit information to the search engine, and receive the results back again and display them to the client.

The server executes CGI programs in order to satisfy particular requests from the browser. The browser expresses its request using hyper text transfer protocol.

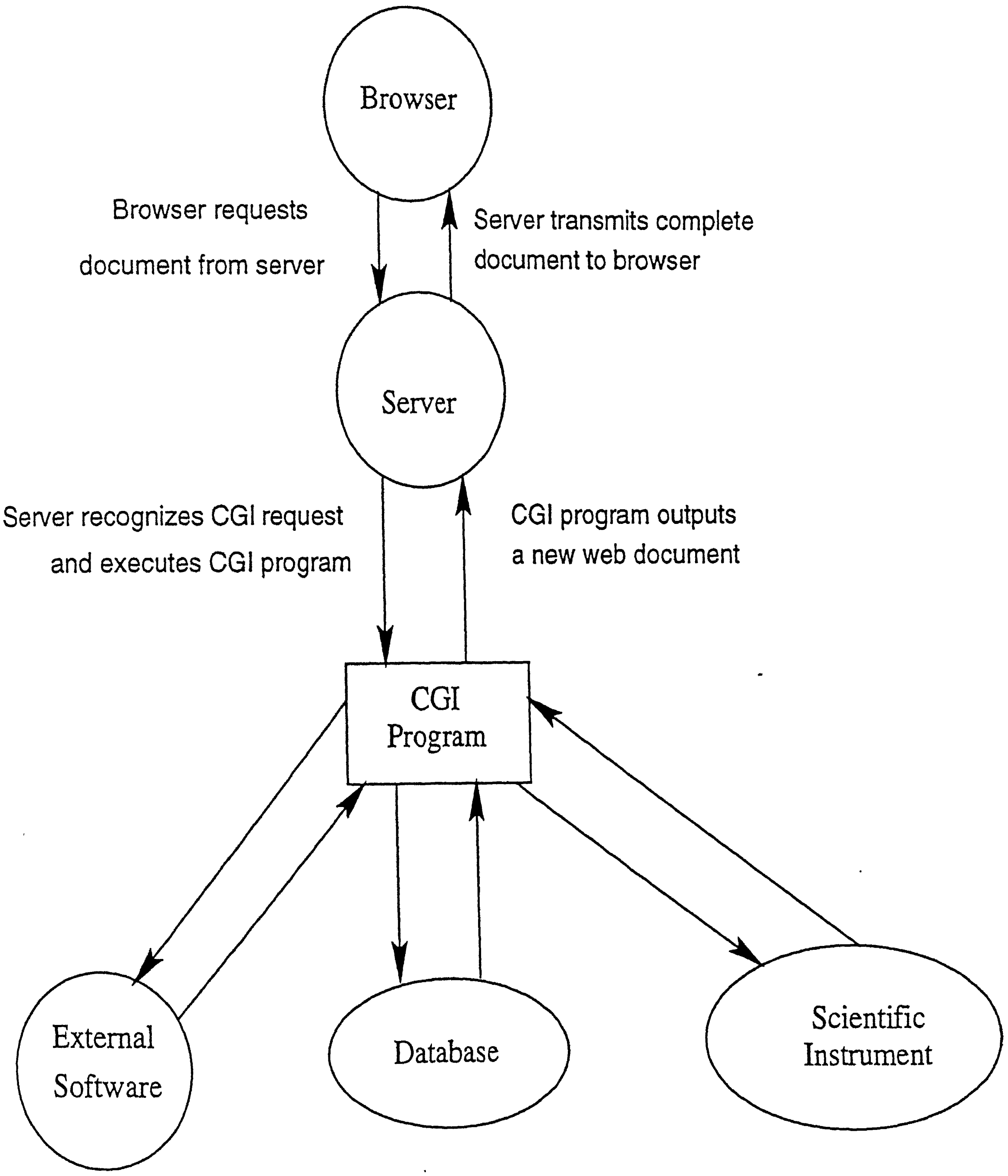

Browser
Browser requests document from server
Server transmits complete document to browser
Server
Server recognizes CGI request and executes CGI program
CGI program outputs a new web document
CGI Program
External Software
Database
Scientific Instrument

Figure 2.1: Communication between browser and search engine

The server responds by returning either document, error status code or a reference to other document.

HTTP requests can be of several types, which are called methods. For CGI programmers, the most important methods are POST and GET.

2.1.1 How a CGI program takes information from the server

CGI programs use environment variable to take their parameters. The two major environment variables used for this purpose are:

Query_string Query string is defined as anything which follows the first ? in the URL. This information could be added either by an ISINDEX document or by a HTML form(with the GET action). This string will usually be an information query, i.e., what the user wants to search for. This string is encoded in the standard URL format of changing spaces to +, and encoding special characters with %xx hexadecimal encoding. Before using it user is required to decode it.

Path_Info CGI allows for extra information to be embedded in the URL. This can be used to transmit extra context specific information to the scripts. This information is usually made available as extra information after the path of the gateway in the URL. This information is not encoded by the server in any way.

2.1.2 How documents are sent back to the client

CGI programs can return many types of documents. They can return image , plain text, HTML document or audio clips to the client. The client must know what kind of document the CGI script is returning. In order to know the document type for a client the CGI program must tell the server what type of document it is returning.

In order to tell the server what kind of document the CGI script is sending back, CGI scripts place a small header before the document. This header is a ASCII text, consisting of the lines separated by either line feeds or carriage returns followed by a single blank line. The output body then follows in whatever native format.

2.2 Process of Searching

The search engine can search the results in a few seconds since it uses an index of the documents instead of scanning the actual documents. The search engine creates the index using an application called indexer.

The search engines create the index in two steps. In the first step they create a list of the documents which are required to index. Some of the search engines like lycos, altavista, do this automatically. They crawl from URL to URL on each site of the web and add each publically available URL to the list. Other search engines like, yahoo depend on each site to send its information to the search engine.

The second step of indexing is scanning each document in the list, extracting the information required for indexing and adding this information to the index. Different search engines index different parts of the documents. Some search engines index only title and headers, while others index full text of the documents.

Quality and quantity of the results retrieved depends on the second step of creating the index. Search engines which index only headers, titles, link labels, give more relevant results while the search engines which index full text of the document, give more comprehensive results.

2.3 Some Popular Techniques for Indexing

2.3.1 Robots, Spiders and Worms

During 1993 many WWW users discovered resources by net-surfing: going to one web server, exploring what was available and then following links to other WWW server. A number of software developers produced softwares which automated this process, so that a program went from server to server, indexing information, such as content of title tag or the content of h1 tags. Such programs came to be known as robots, spiders or worms[9]. Most of the search engines like lycos,wecrowler etc. use this technique for indexing.

There are a number of problems with this approach to indexing:

Server performance: When a robot arrives at your server it can place a heavy load on the server.

Network performance: Robots can place a heavy load on the network.

Maintenance: The robot will not know if an information source that has been indexed is withdrawn from service.

2.3.2 Aliweb

Aliweb(Archie Like Indexing In The Web)[17] provides another approach to the indexing of the WWW resources. In Aliweb each site is responsible for indexing files. The server administrator is responsible for choosing the files to be indexed.

2.3.3 Swish

Swish[18] which stands for simple web indexing system for humans, was announced in 16 November 1994. It is a program that allows you to index your web site and search for files using keywords in a fast and easy manner.

2.3.4 Wais

Wais(Wide area information server)[16] is another mechanism for indexing resources. Wais is used by the Computing service, University of Leeds to index its documents and news letters.
A detailed information about search engines can be found at [7,8].
These approaches to searching have few disadvantages, which are as follows:

1. Quality of User interface is very poor.
2. Query language can specify only content of the document, structure of the document can not be specified.
3. Size of the index gets very large.
4. Periodically re-indexing is necessary, to keep the system consistent.

5. Instead of giving the relevant part of the target document whole document is returned.

For intranets where size of the data to be searched is limited, a different approach can be used for searching the documents. In this approach documents are searched exhaustively and no pre-indexing of the documents is required. Since the documents are searched exhaustively content as well as the structure of the document can be specified in the query.

Chapter 3

A New Approach for Searching HTML Documents

3.1 Overview

Query language supported by traditional search engines is very elementary. User can specify content of the document, but he can not specify the structure of the document. For arbitrarily formatted documents query language can not specify the structure of the document. But, if the documents are formatted according to some pre-specified rules, we can specify content as well as the structure of the documents in the query. These types of queries would be more powerful comparative to content based queries.

Another problem with the traditional approach is that user can not specify what part of the document is to be retrieved(displayed). Even though a user wants to see a section containing some specific key-words, he is forced to view the whole document. A powerful query language should be able to specify which part of the document is required.

TQHT is a new approach for searching the HTML documents. In this approach documents are searched exhaustively and no pre-indexing is required. TQHT reads the document and then takes the decision based on the content and the structure of the document. User can specify his requirements more precisely, using it.

In this approach user is also able to specify which part of the document should be returned. TQHT breaks the document into parts specified by the user. After breaking down the document in parts, TQHT evaluates the query on each part. Parts which matches this query are returned as the result of the query.

In this approach pattern matching is also supported, users can use regular expressions instead of exact strings.

One interesting fact is that, humans have a very sharp visual memory. They can easily remember that they have seen a word *search engine* in bold font, and a word *indexer* in italic, near by.

3.2 Text Modeling

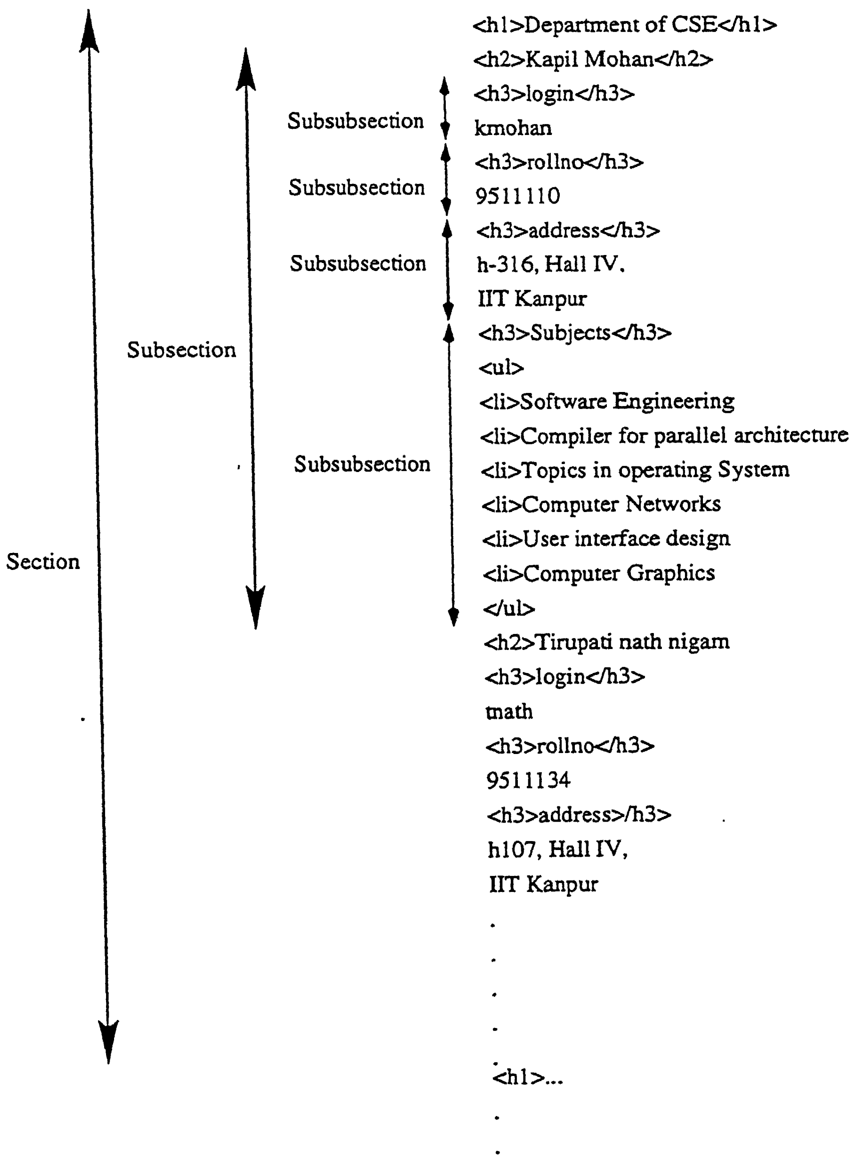
In TQHT text is modeled as hierarchical sections. Each HTML document is divided into a number of sections, each section is further subdivided into a number of subsections and so on. Basic unit of the text is para.

Each part of the document starting from **h1** level header is considered as a section. Each part of the document starting from **h2** level header and ending on **h1** or **h2** is considered as a subsection. Other hierarchies of the document structure are decided on the basis of the **h3,h4,h5,h6** and p tags. In the figure 3.1 a sample HTML document with its hierarchies is shown.

Using this text model user can specify which part of the document is required. user can specify whole document, section of the document, subsection of the document or other parts.

3.3 Architecture of the System

Architecture of TQHT is shown in the figure 3.2. TQHT has two parts, client-applet and server application. The server application is the back-end of TQHT, while the client applet is the front-end of TQHT. The server application reads the actual documents and searches the results of the query. The client applet receives the query, passes this query to the server application, and displays the query results



```
<h1>Department of CSE</h1>
<h2>Kapil Mohan</h2>
<h3>login</h3>
kmohan
<h3>rollno</h3>
9511110
<h3>address</h3>
h-316, Hall IV.
IIT Kanpur
<h3>Subjects</h3>
<ul>
<li>Software Engineering
<li>Compiler for parallel architecture
<li>Topics in operating System
<li>Computer Networks
<li>User interface design
<li>Computer Graphics
</ul>
<h2>Tirupati nath nigam
<h3>login</h3>
tnath
<h3>rollno</h3>
9511134
<h3>address>/h3>
h107, Hall IV,
IIT Kanpur
.
.
.
.
.
<h1>...
.
.
```

Figure 3.1: Structure of a HTML document

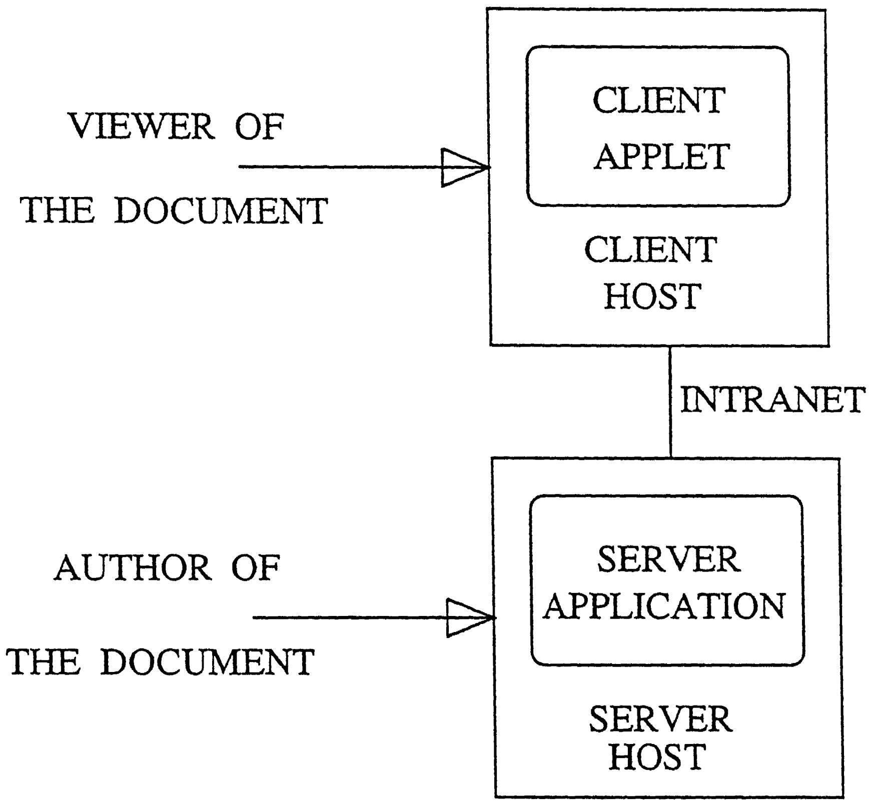


Figure 3.2: Architecture of the system

retrieved from the server application. We have used sockets to communicate between the server application and the client applet.

3.3.1 Client Applet

The client applet handles the interaction part. It works as a gateway between the user and the server application. Author of the HTML document can add the client applet any where in the document. At the time of viewing the HTML page, the client applet initializes itself depending on the parameters. The client applet takes six parameters **select**, **from**, **where**, **label**, **type** and **frame**. (These parameters are discussed in the appendix B.) After initializing, it waits for user interaction, and performs various operations according to user need. (These operations are discussed in the section 3.4)

3.3.2 Server Application

The server application is responsible for searching the HTML documents and finding out the results based on the query. The server application is started before the client applet sends any request. The server application opens a socket and waits for the request. Whenever a request is received by the server application, a new thread is created by the server application to handle that request. This thread searches the results and returns them to the client applet.

3.4 User Interface of the Client Applet

Interface of the client applet is shown in the figures 3.3 and 3.4. There are two interfaces shown, first for entering the query and second for viewing the results.

3.4.1 Interface for Entering the Query

This interface has three text fields: **select**, **from** and **where**; and one **submit** button. User can set query by filling these fields and pressing the **submit** button. After setting the query, applet changes its interface to the second one.

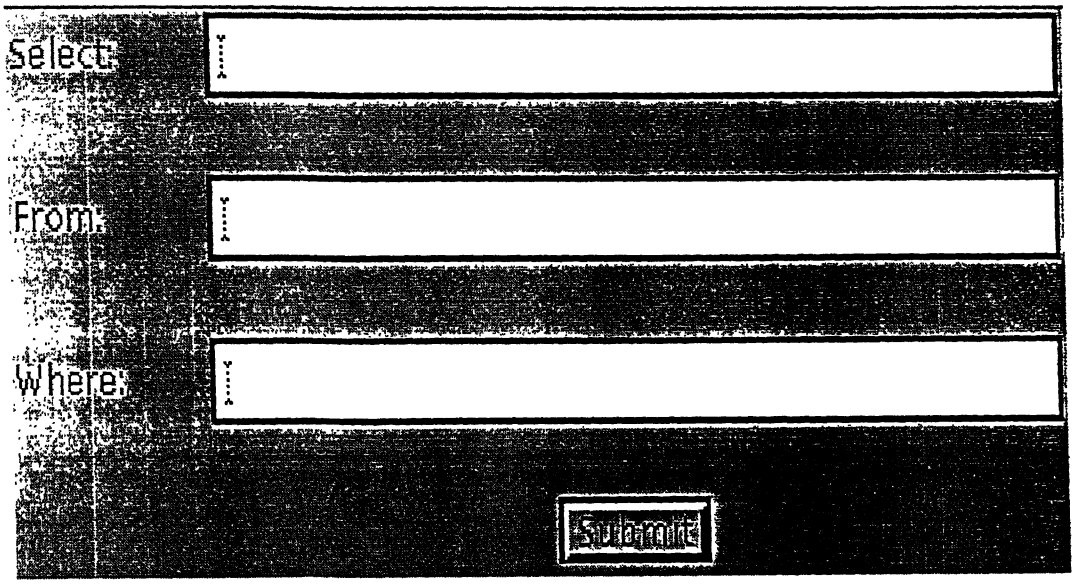


Figure 3.3: Interface for entering query

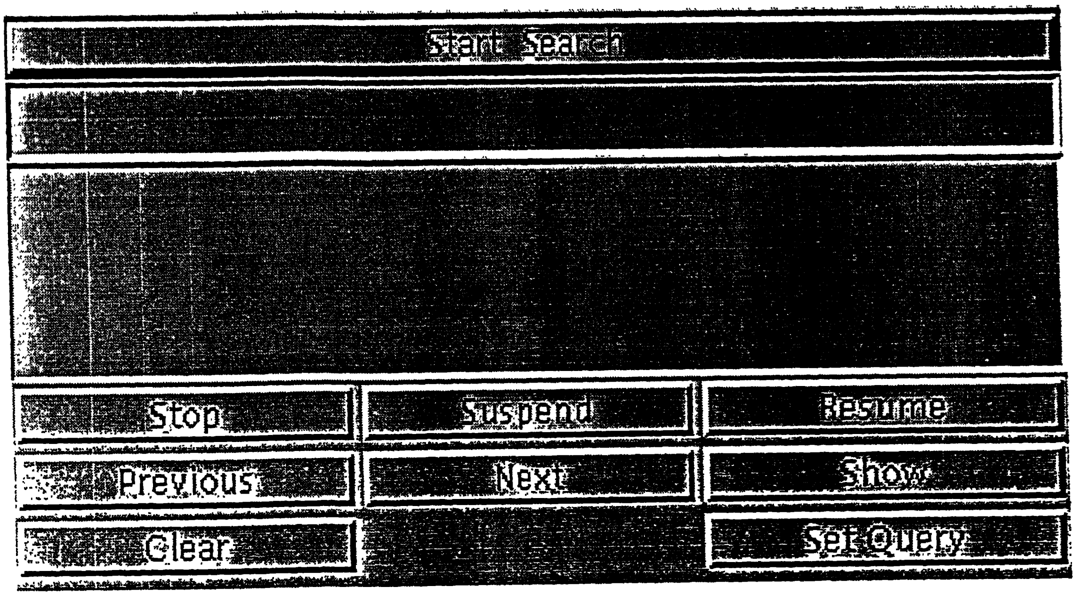


Figure 3.4: Interface for viewing results

3.4.2 Interface for Viewing the Results

In this interface there is a button on the top of the applet. Label of this button is set to the value of the parameter **label**, which is set by the author of the document. User can start search by pressing this button. Below the button there is a text field for messages. Below the text field there is a selectable list. One line of each result is added to the selectable list. Below the list there are 8 buttons for controlling the applet and the server. User can view the content of any result by selecting it and pressing **show** button. Others buttons can also be used for controlling the server and viewing the results.

Functions of the buttons are as follows:

STOP - This button stops the server from further searching the results.

SUSPEND - This button temporarily stops the server from further calculating the results.

RESUME - This button re-starts the server, which was temporarily stopped by pressing the SUSPEND button. The server starts searching from the point where it stopped last time.

SHOW - This button shows the result, currently selected in the selectable list.

PREVIOUS - This button shows the previous result, if any.

NEXT - This button shows the next result, if any.

CLEAR - This button clears the selectable list, message field and the window for showing the results.

SET QUERY - This button changes the interface to first one, now user can change the query.

TQHT works in two modes (search engine or dynamic link) depending on the value of the parameter **type**, which is set by the author of the HTML document. If the author of the document wants the client applet to be used as a search engine then he sets the value of the parameter **type** to *se*, and if author wants the client applet to be used as a dynamic link then he sets the value of parameter **type** equal to *dl*. We will discuss these interfaces for each mode separately.

▪ *When parameter* **type** *is se (search engine)*

The interface for setting the query is shown when applet initializes. As soon as the user sets the query, interface changes to the other one. User can view the results using control buttons.

▪ *When parameter* **type** *is dl (dynamic link)*

The interface for viewing the results is shown when the applet initializes. Author of the document sets the query, so first interface is not required. User can view the query results using the control buttons.

3.5 Specification of the Query Language

TQHT can understand the HTML document structure, so queries can be based on content as well as structure of the HTML documents.

A query has three parts -

- select
- from
- where

A combination of these three parameters forms a query. **From** gives the locations to search, while **select** and **where** represent what to search. A document is considered as a collection of units. For example if **select** is *h2* then TQHT considers the documents as a collection of units, where each unit starts from *h2* and ends at *h1* or *h2*. Each unit of the document which matches the **where** is selected.
Now we will describe all the three parts of the query in detail:

3.5.1 select

This part represents which part of the document is to be selected. Select can have following values -

h1,h2,h3,h4,h5 or h6, h*,h-1,h-2,h-3,h-4,h-5,ul,ol,menu,blockquote,para

We can divide these tags into three categories.

h1,h2,h3,h4,h5 or h6 - If **select** is *h1* then whole section is selected. If **select** is *h2* then subsection is selected and so on.

h*,h-1,h-2,h-3,h-4 or h-5 - In this case any section, subsection, subsubsection etc., which matches the **where** may be selected. Lowest hierarchical unit having the specified information is selected. If we want to select unit hierarchically one level up, we can give the **select** as *h-1*, for two level up *h-2* and so on. For example if **select** is *h-1* and **where** matches with a subsubsection then subsection, which contains that subsubsection is returned as the result of the query. The highest hierarchical unit is whole document. TQHT returns the whole document if hierarchical level specified by user is not present in the document.

ul,ol,menu,blockquote or para - Other possible values of **select** are ol(ordered list), ul(unordered list), menu, blockquote, para. If **select** is *ol* then each ordered list in the HTML document can be selected, remaining parts are ignored. Similar is the case with **ul, menu** and **blockquote.** If the **select** equals *para* then para of the HTML document(which is decided based on **p** tags and headers) is selected.

3.5.2 from

This part represents from which documents search should start. The server searches all the documents specified in the source as well as the documents which are linked directly or indirectly from the source documents. So performance of TQHT depends on the specification of source. Initially, all the source documents are added to a list, called **source_loc.** Now each document in this list is searched and as soon as any link in the document appears, address of the link is added to the list (if this address is not already present in this list). TQHT checks whether the link is absolute or relative, if the link is absolute then link is added to the list as it is, otherwise TQHT adds the base address of the current document to this link and adds this address to the list.

User can specify regular expressions for the source. In this case TQHT matches these regular expressions in a pre-specified list of URLs, and adds all the matches to the **source_loc** list. Order of search is breadth-first search.

3.5.3 where

This part represents actual query string. Using this string server decides which of the units can be selected. Server matches the **where** with each unit, and the unit which matches the **where**, is returned as the result of this query.

Where is a query expression(exp) syntax of which can be represented using following grammar:

```
exp    - (exp OR exp) | (exp AND exp) | sexp
sexp   - (tag : ptm)
tag    - any HTML tag | h* | txt
ptm    - ptm OR ptm | ptm AND ptm | regexp
regexp - any global type regular expression.
```

Query is a collections of **sexp's** connected with boolean operators(AND,OR). Each **sexp** represents which strings(or regular expressions) should be present in which part of the document. Each **sexp** is a sub-query, it has two parts **tag** and **ptm**(patterns to match), **tag** can be equal to, any HTML tag, *h** or *txt*, and **ptm** is a string of many regular-expressions connected with AND and OR. Value of **sexp** is decided for each unit of the document and it is set to one if the corresponding **ptm** is present in the corresponding **tag** in the unit.

Here is a sample of sexp to make it more clear

```
h2 : intro* OR summary
```

In this case value of the **sexp** will be equal to one if either *intro** or *summary* regular expressions matches with the string present in **h2** level heading, else value of this **sexp** will be equal to 0.

Value of the **exp** is decided in the following way.

exp can contain any number of **sexp** connected with OR and AND. Parenthesis can also be used. Now for each unit, value of each **sexp** is decided, as above described. After deciding value of each **sexp** value of the **exp** is evaluated. If value of the **exp** is evaluated equal to one then this unit is added to the result of the query, otherwise this unit is rejected.

If value of the **tag** is *h** then **ptm** can appear in any header, and if value of tag is *txt* then **ptm** can be present any where in the unit.

3.6 Pattern Matching

One can specify regular expressions instead of exact strings, in the **from** and the **where** part of the query. Instead of giving exact source locations one can specify a regular expression. All the URL's matching this regular expression are considered as **from**.

In the **where** part one can use regular expressions in the **ptm**. For the pattern to match a string each character of the pattern must be same as the corresponding character of the string, except that a few characters in the pattern can be interpreted specially. For example a * in the pattern matches a substring of any length, so *java** matches any string whose first four characters are *java*. Here is a list of all the special characters supported in the pattern matching:

* Matches any sequence of zero or more characters.

? Matches any single character.

[**chars**] Matches any single character in chars. If chars contains a sequence of the form a-b, any character between a and b, inclusive will, match.

\x Matches the single character x. This provides a way to avoid special interpretation for any of the characters *?[] in the pattern.

Chapter 4

Implementation Details

In the present work a tool for querying HTML documents is implemented. This chapter presents the various details about the implementation. In the section 4.1 reason behind the choice of **java** is presented. In the section 4.2 different classes and their relationships are discussed. Section 4.3 discusses protocol for communication between the client applet and the server application. Section 4.4 discusses the need for the qthread.

4.1 Choice of Java

We have used **java** as the language for implementing TQHT because TQHT requires following capabilities in the host language:

Threads - Server should be able to handle each query request in parallel. Some mechanism like thread is required to support it. In **java** threads are supported.

Portability - For making the server application portable across multiple platforms host language should be portable. **Java** supports this feature.

Concept of applet - **Java** has a concept of applet. Applet is a program which can be down loaded across the network and a java enabled web browser can show it. Applet provides a good quality of interface, which is not possible with the form based interfaces. The java.awt package has almost all the components required for a good user interface.

Capability of lexical analyzer - Package java.util has a class stringtokenizer which can tokenize any string.

4.2 Major Classes

Major classes used in the implementation of TQHT are as follows:

Class aplt

```
public class aplt extends Applet {
String select;
String from;
String where;
String label;
String type;
String frame;

public void init() {};
public boolean  action(Event e,Object o) {};
}
```

Class serv

```
public class serv {
public static void main(String args[]) {};
public static void server() {};
}
```

Class qthread

```
public class qthread extends Thread{
int port;

public void  run() {};
}
```

Class regexp

```
public class regexp {
  regexp(String re) {};
  public int re_match(String str) {};
}
```

Class matchfile

```
public class matchfile {
  matchfile(String select,String from,String where,int opf) {};
  public void start_search(InputStream is,PrintStream dos) {};
  public void match(String filetp) {};
}
```

Class slist

```
  public class slist {
  public static String[] srclist(String src,rInt cnt) {};
}
```

Class rInt

```
public class rInt {
public int intv;
}
```

Class Cf

```
public class Cf {
 public PrintStream prs;
 public int start_line;
 public int start_col;
 public int end_line;
 public int end_col;
 public int lci;
 public int unit;
 public String file_name;
 Cf(int portno) {};
 public  void append_to_result() {};
 }
```

Class par

```
public class par {
public  String source_loc[];
public byte exp_position[];
public byte q_array[];
public regexp re_array[];
public byte exp_type[];
public int max_brace_level;
public int q_array_length;
public int exp_length;
public int source_loc_count;
public String unit;


par(String select,String from,String where) {};
public  void add_to_list(String str) {};
}
```

The **aplt** class implements the client applet, while all other classes implements server application. First we will discuss **aplt, serv** and **qthread** classes. Figure 4.1 shows relationship between these classes.

The **aplt** class handles the user interface part of the TQHT. The **serv** class works as a daemon process for processing the query. The **aplt** class takes all the three parts of the query either as the applet parameters or through the user interface and stores these values in the variables **select, from** and **where.** When user presses the button **start-search**, the **aplt** class creates a socket for communicating with the **serv** class, and sends a request to the **serv** class. The **serv** class creates a new thread of the class **qthread**, and assigns a port number to this class. The **serv** class sends the port number assigned to the **qthread** to the **aplt** class. Now the **aplt** class creates a socket for communicating with the **qthread** class and sends the query to the **qthread** class. The **qthread** class evaluates the results and sends them to the **aplt** class. Now the **aplt** class shows all the results to the user.

Port number passed to the **qthread** class is an integer number. The **serv** class increments this number after creating each **qthread** class, to avoid clashes in the port number.

The **qthread** class takes help of the following classes in the process of searching the results:

regexp This class is used for matching the regular expressions. At the time of creating a instance of this class a regular expression is passed to the constructor. The **regexp** class parses the regular expression and creates a Finite State Automata(FSA), which represents this regular expression. This class

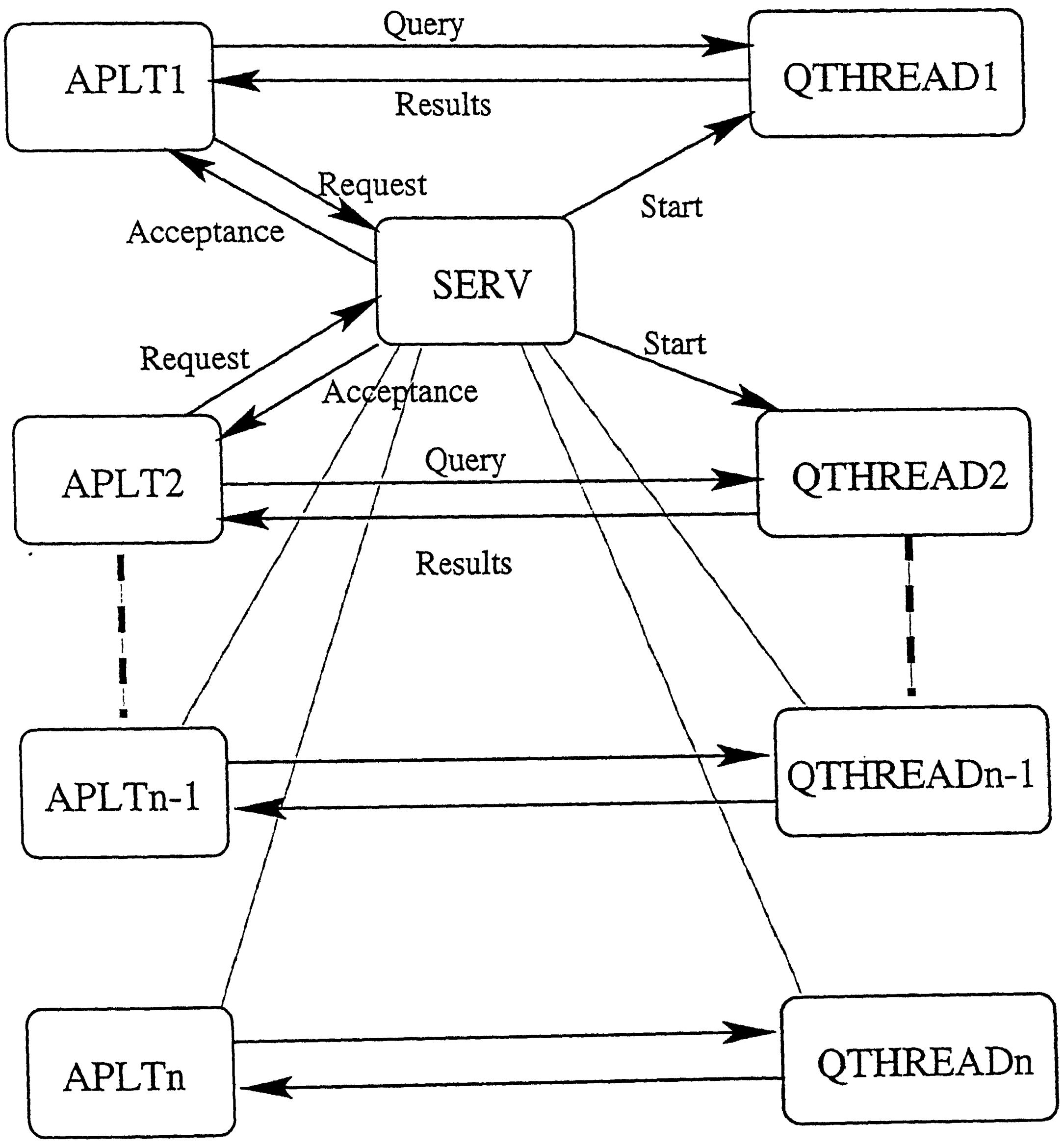


Figure 4.1: Communication between aplt, serv and qthread classes

has a method **re_match**, which takes a string and matches this string with the regular expression specified at the time of instantiation. The **re_match** method returns one if the string matches with the regular expression, otherwise it returns zero.

par This class is responsible for parsing the query and initializing the data structures required at the time of evaluating the query. This class takes **select, from** and **where** as parameters at the time of instantiating. This class creates a list of regular expressions required.

matchfile This is the most important class. This class is responsible for scanning the files and finding out the results. The **match** method of this class scans each file in the list **source_loc** from top to bottom and for each unit evaluates the query, as soon as any unit matches the query, the **match** method calls the **append_to_result** method of the class **Cf**. The **append_to_result** method of the Cf class creates a separates file containing this unit and passes the address of this file to the client applet. At the time of parsing a file, if the **match** method comes across any link to another file then the **match** method calls the **add_to_list** method of the class **par**. This procedure takes the file address(URL) as a parameter and adds this file address to the list **source_loc** if it is not already present in it.

Cf This class is responsible for creating the results. The **match** method of the **matchfile** class passes the **file_name** and starting and ending locations of the matching part to it. This class creates a new file containing specified information and sends the address of the new file to the **aplt** class.

slist This class has a static method **srclist** which takes the **source** part of the query as parameter and returns all the URLs matching with this specification. This method matches all the specifications in the **source** with a prespecified list of URLs and returns a list of all the matches.

4.3 Protocol for Communication

The **aplt** class and the **serv** class use a pre-specified protocol for communication. Initially the **serv** class opens a server socket on the pre specified port and waits for some request. Now the **aplt** class creates a socket for communicating with **serv** class and sends a string *requestq* to the serv class. The **serv** class reads this string and decides that one **aplt** class is requesting for processing a query. The **serv** class starts a new **qthread**, and sends port number assigned to qthread to the **aplt** class, and waits for the next request. Now the **aplt** class sends **select, from** and **where** strings followed by their values to the **qthread**. The **qthread** reads these strings and starts search by calling **start_search** procedure of the class **matchfile**. After scanning each file **start_search** checks for message from the **aplt** class. Three types of messages are possible : stop, suspend and resume. If message(string received) is **stop** then **start_search** returns and the **qthread** stops. If the message is **suspend** then **start_search** procedure waits for another message. If another message is **resume** then **start_search** starts search from the next file in the array **source_loc**, and if the message is stop it causes to stop the **qthread**.

4.4 Need for Qthread

Class **qthread** extends thread class. The **serv** starts a new thread each time a request comes. The **qthread** class handles that request independent of the server, so server can accept a new request. In this way all the requests are processed simultaneously. The **serv** could process all the request one by one sequentially itself. But in that case performance of the server would be very poor.

Chapter 5

Conclusion and Future Work

We have presented a system(TQHT) which allows queries based on content as well as structure of the document. This system searches the content of the actual document, instead of using an index. In an intranet environment where time taken for downloading a document is negligible, and the number of documents is not very large, this approach for searching the text can be quite useful. TQHT can be used as a dynamic link or as a search engine

5.1 As a Dynamic Link

TQHT can by customized to work as a dynamic link. We can add the client applet at the place where we want to create the dynamic link. Dynamic links can be useful in the applications like multi purpose text book[8], and in a hypermedia system which is evolving.

5.2 As a Search Engine

TQHT can be used as a powerful search engine for small networks. It will eliminate the use of the indexer application.

5.3 Limitations

1. Speed of searching the documents is slower, comparative to traditional search engines.

2. Only a limited number of sites can be searched.

3. A person, who does not know HTML, can not use TQHT effectively.

5.4 Future Work

1. Work can be done to scale this tool for Internet. Since each server can handle a small area of the Internet, to cover a large part of the Internet such servers can co-operatively work together. For realizing such a system we need a protocol for partitioning the Internet and collect the results in a hierarchical way.

2. TQHT can be extended to query other document styles also like Latex. Database support can also be incorporated in TQHT.

3. We can also incorporate some sort of cache mechanism in TQHT, so that frequently asked queries can be searched faster.

References

[1] Golzalo Navarro , Richardo Baeza-Yates. A language for queries on Structure and content of textual databases. SIGIR July 1995, Page:93-101

[2] Andrew M. Fountain et. al.. MICROCOSM: An open model for Hypermedia with dynamic linking. Hypertext: INRIA 1990, Page:298-311.

[3] Helen Ashman, Janet Verbyla. Dynamic link management via the functional model of the link.
http://www.cs.flinders.edu.au/research/hypermedia/biwit94.html

[4] Helen Ashman, Janet Verbyla. Dynamic and externalized linking: Requirements of a large-scale hypermedia.
http://www.cs.flinders.edu.au/research/hypermedia/nt93workshop.html

[5] Licia Calvi, Paul De Bra. Using dynamic hypertext to create multi-purpose textbooks.
http://wwwis.win.tue.ne/ debra/ed-media/paper.html

[6] D. Jaya Ram. A hypertext system with programmable links.
M.Tech Thesis, MT-CS-90-12 IIT Kanpur, March 1990.

[7] Virginia M. Heinrich, Ronald R. Hauser, Mary M. Moran. Searching the Internet tutorial.
http://www.mlrc.stthomas.edu/profdev/library/home.htm

[8] Gus Venditto. Search engine Showdown.
http://www.internetworld.com/1996/05/showdown.html

[9] The web robots page.
http://info.webcrawler.com/mak/projects/robots/robots.html

[10] A list of browsers.
http://www.w3.org/hypertext/WWW/clients.html

[11] A comparison of browsers.
http://www.osf.org/ kiniry/projects/web/browser_comparison.html

[12] A beginners guide to HTML.
http://www.ncsa.uiuc.edu/demoweb/html-primer.html

[13] A list of HTML tools.
http://www.w3.org/hypertext/WWW/Tools/Filters.html

[14] A list of search engines.
http://cui.unige.ch/meta-index.html

[15] T. Rama Kumar. HYSOMA: A Hypertext Based CASE tool for C software maintenance.
M.Tech Thesis,MT-CS-95-07,IIT Kanpur, Feb95

[16] A WAIS overview.
http://www.w3.org/hypertext/Products/wais/sources/Overview.html

[17] ALIWEB - Archie like indexing in the web.
http://web.nexor.co.uk/mak/doc/aliweb-paper/paper.html

[18] Documentation of SWISH.
http://www.eit.com/software/swish/swish.html

[19] Thomas Boutell, CGI programming in C and Perl.
Addison-Wesley Publishing Company.

[20] CERN Home Page.
http://www.cern.ch/

Appendix A

Some query examples

- Suppose information about books is organized publisher wise, One HTML document contains a list of TMH books, one a list of BPB books ... . Now if a user wants to see all the books on *java*, independent of the publisher of the book he can give the following query.

```
select = h*
from = http://www.tmh.org/list.html;http://www.bpb.org/lob.html;..
where = (h* : java)
```

- Suppose some one saw a document having a word *IIT* in italic and word *library* in bold nearby. For retrieving the same document following query can be given

```
select = h*
from = http://144.16.162.33/
where = (bold : library) AND (italic : IIT)
```

Appendix B

User Manual

There are two types of the users of this tool

- Author of the HTML document.
- Viewer of the HTML document.

Author of the HTML document is responsible for including the client applet(aplt.class) in the HTML page, setting the values of parameters, creating frames for showing the applet and for showing the results of the query. Viewer of the HTML document should know how to set the query, how to start search and how to see the results retrieved as a result of query.

B.1 Author of the document

Author of the HTML document should follow following steps for embedding the client applet in his HTML page.

1. Create a HTML file containing only the client applet. Let the name of the file is *show1.html*. Set all the parameters of the client applet as required. Value of the parameter **frame** depends on the name of the frame created for showing the results, in the file *show.html*.

2. Create another file(*show.html*) having two frames. In one frame show the *show1.html*. Second frame is reserved for the results. Name of the second

frame should be passed as **frame** parameter to the applet *aplt.class* in the file *samp1.html*

A list of parameters is as follows:

select - This parameter is *select* part of the query.

from - This parameter is *from* part of the query.

where - This parameter is *where* part of the query.

label - This parameter is the label of the button which starts the search. It should reflect the theme of results of query.

type - This parameter decides how the client applet will work, as a dynamic link or as a search engine. If the author wants the client applet to be used as a dynamic link then he should set this parameter to *dl*, and if the author wants the client applet to be used as a search engine then he should set this value to *se*.

frame - This parameter passes the name of the frame, in which results will be displayed.

Sample examples of *show.html* and *show1.html* is given below

```
<HTML>
<head>
<title> show.html </title>
<FRAMESET ROWS="50%,50%">
<FRAME SRC=show1.html NAME="frame_for_aplt">
<FRAME SRC=help.html NAME="frame_for_result">
</FRAMESET>
</head>
<body>
This line will not be visible
</body>
</HTML>
```

```
<HTML>
<head>
<title> show1.html </title>
</head>
<body>
<APPLET CODE="aplt.class" WIDTH=500 HEIGHT=200>
<PARAM NAME=select VALUE="h2">
<PARAM NAME=from VALUE="samp.html">
<PARAM NAME=where VALUE="ul : User inter*">
<PARAM NAME=label VALUE="All the students">
<PARAM NAME=type VALUE="dl">
<PARAM NAME=frame VALUE="frame_for_result">
</APPLET>
<hr>
<h2> A  search  engine  for  Intranets</h2>
</body>
</HTML>
```

help.html will be shown in the window before starting the search, so author of the document can put any useful information in this file.

B.2 Viewer of the document

Viewer of the HTML document is required to handle the client applet in different ways for different types. We will discuss each type:

B.2.1 When the client applet is working as a search engine

In this case user views the interface for entering the querying when he views the HTML document containing the client applet. User is required to enter the values of **select**, **from** and **where**. After entering the values of these fields user should press the **submit** button. A screen shot of this process is shown in the figure b.1.

As soon as user presses the **submit** button interface changes to the another one(for showing the results).

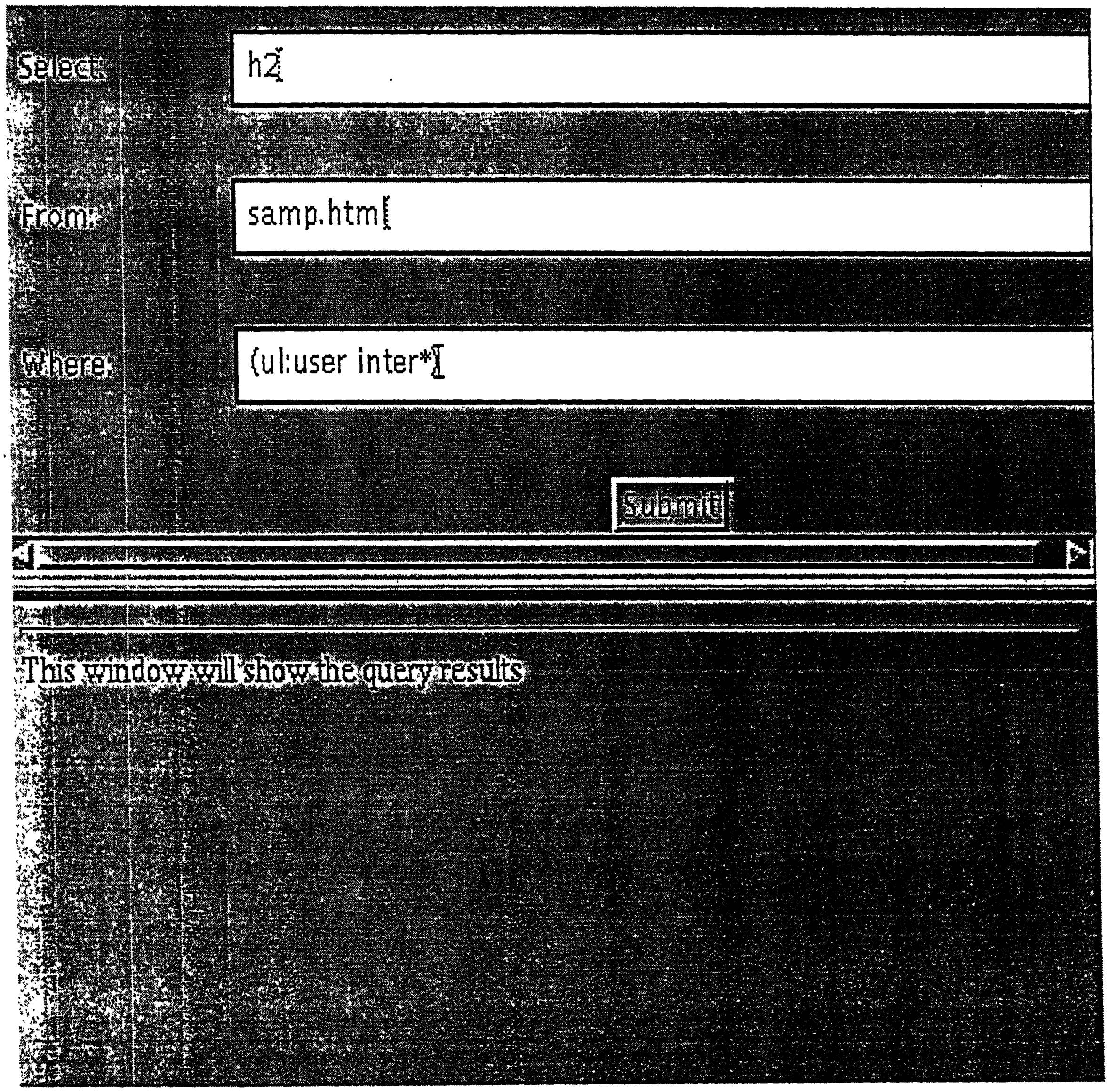
For starting the search user should press the button **Start search** After pressing the button message window should show the message *Server contacted Please wait* or *Connection Failed: Server is not running.* If the message is *Connection failed : Server is not running* then server is not running and user should ask the system administrator to run the server.

If the message is *Server contacted : Please Wait* then the client applet has sent the request to the server application and the server application has started the searching. Now user should wait for the results. As soon as the result will come it will be added to the selectable list of the client applet and contents of the result will be displayed in the window specified for results. A sample shot of this state is shown in the figure b.2.

Now user can use different buttons for displaying next or previous results. User can use stop,suspend and resume buttons for controlling the server.

B.2.2 When the client applet is working as a dynamic link

In this case user views the interface for viewing the results, user can not change the query parameters. As soon as the viewer presses the button on the top, search starts. Remaining procedure is same as in search engine.



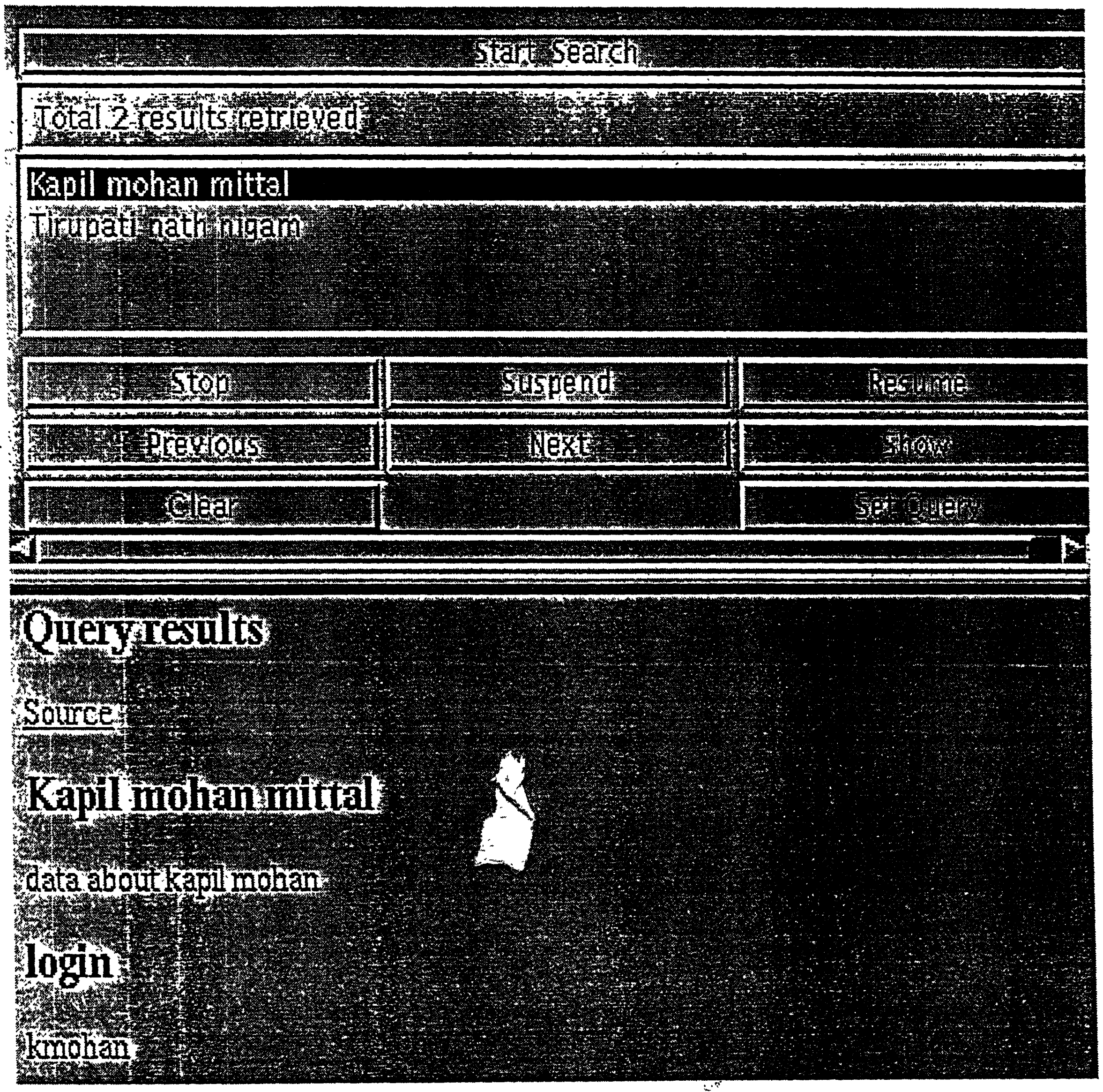
h2

samp.htm

(ul:user inter*

This window will show the query results

Figure B.1: A sample query



Total 2 results retrieved

Kapil mohan mittal
Tirupati nath nigam

Query results

Source

Kapil mohan mittal

data about kapil mohan

login

kmohan

Figure B.2: Results of the query